Journal of Computer and System Sciences 76 (2010) 428–446

Contents lists available at ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

Scaling up workflow-based applications

Scott Callaghan [a], Ewa Deelman [b,*], Dan Gunter [e], Gideon Juve [a], Philip Maechling [a], Christopher Brooks [f], Karan Vahi [b], Kevin Milner [a], Robert Graves [c], Edward Field [d], David Okaya [a], Thomas Jordan [a]

[a] University of Southern California, Los Angeles, CA 90089, United States
[b] USC Information Sciences Institute, Marina Del Rey, CA 90292, United States
[c] URS Corporation, Pasadena, CA 91101, United States
[d] US Geological Survey, Pasadena, CA 91106, United States
[e] Lawrence Berkeley National Laboratory, Berkeley, CA 94720, United States
[f] University of San Francisco, CA 94117, United States

ARTICLE INFO

Article history:
Received 18 March 2009
Received in revised form 22 August 2009
Available online 22 November 2009

Keywords:
Scientific workflows
Distributed applications
Workflow scalability

ABSTRACT

Scientific applications, often expressed as workflows are making use of large-scale national cyberinfrastructure to explore the behavior of systems, search for phenomena in large-scale data, and to conduct many other scientific endeavors. As the complexity of the systems being studied grows and as the data set sizes increase, the scale of the computational workflows increases as well. In some cases, workflows now have hundreds of thousands of individual tasks. Managing such scale is difficult from the point of view of workflow description, execution, and analysis. In this paper, we describe the challenges faced by workflow management and performance analysis systems when dealing with an earthquake science application, CyberShake, executing on the TeraGrid. The scientific goal of the SCEC CyberShake project is to calculate probabilistic seismic hazard curves for sites in Southern California. For each site of interest, the CyberShake platform includes two large-scale MPI calculations and approximately 840,000 embarrassingly parallel post-processing jobs. In this paper, we show how we approach the scalability challenges in our workflow management and log mining systems.

1. Introduction

Many scientific applications today are being structured as scientific workflows [1,2]. Scientific workflows describe the relationship of their individual computational components and their input and output data in a declarative way. In astronomy, scientists are using workflows to generate science-grade mosaics of the sky [3–5], to examine the structure of galaxies [6] and study other phenomena. In bioinformatics, they are using workflows to understand the underpinnings of complex diseases [7,8]. In earthquake science, workflows are used to predict the magnitude of earthquakes within a geographic area over a period of time [9–11]. In physics, workflows are used to try to measure gravitational waves [12] and model the structure of atoms [13]. In ecology, scientists explore the issues of biodiversity [14].

Scientific workflows enable the formalization of the computations, exposing individual computational steps, and data and control dependencies between them. They allow scientists to focus on the logical structure of the overall computation

* Corresponding author.
E-mail addresses: scottcal@usc.edu (S. Callaghan), deelman@isi.edu (E. Deelman), dkgunter@lbl.gov (D. Gunter), juve@usc.edu (G. Juve), maechlin@usc.edu (P. Maechling), cbrooks@cs.usfca.edu (C. Brooks), vahi@isi.edu (K. Vahi), kmilner@usc.edu (K. Milner), Robert_Graves@URSCorp.com (R. Graves), field@caltech.edu (E. Field), okaya@usc.edu (D. Okaya), tjordan@usc.edu (T. Jordan).

0022-0000/$ – see front matter
doi:10.1016/j.jcss.2009.11.005

rather than on the low-level implementation details. This structure also makes updating and maintaining a workflow-based application easier than modifying a complex script that represents the same computation.

Today, workflow-based scientific applications, often because of their scale, either in the number of processing steps or the size of the data they access, are running on the national and international cyberinfrastructure such as the Open Science Grid (OSG) [15], the TeraGrid [16], EGEE [17], and others. One such application and the focus of this paper is SCEC's CyberShake. Although we illustrate the techniques in the context of CyberShake, our solutions are broadly applicable.

The Southern California Earthquake Center (SCEC) [18] project is using high-performance computing to advance their program of earthquake system science research. As a part of this research program, SCEC scientists have developed a new technique in which full 3D waveform modeling is used in probabilistic seismic hazard analysis calculations (PSHA). However, the complexity and scale of the required calculations prevented actual implementation of this new approach to PSHA. Since that time, SCEC computer scientists have gathered the seismological processing codes and the cyberinfrastructure tools required for this research. The codes and tools have been assembled into what is called the CyberShake computational platform [9,10]. In SCEC terminology, a computational platform is a vertically integrated and well-validated collection of hardware, software, and people that can produce a useful research result. The CyberShake computational platform is designed to perform physics-based PSHA calculations for geographic sites in the Southern California region.

In order to obtain meaningful results, a single execution of a CyberShake workflow contains approximately 800,000 individual tasks. As a result, this application provides challenges in scaling up the workflow description, the workflow execution, and the workflow execution analysis. In this paper, we describe our approach to all three problems, with particular emphasis on the latter two. We discuss how scientists are describing their workflows in a scalable fashion. We give an overview of the Pegasus Workflow Management System used to support the workflow mapping and execution, and focus on the optimizations developed and used to achieve a scalable solution. Finally, we describe the integration of the workflow management system with a log mining tool, NetLogger, that analyzes the performance of large workflows in a scalable manner. Although SCEC has particularly large workflows, the techniques presented here are applicable to other applications with smaller numbers of tasks.

Our results summarize nine science runs of CyberShake performed over a period of three weeks, and we demonstrate the combined Pegasus and Netlogger capabilities applied to the performance analysis of one particular execution of CyberShake.

The paper is organized as follows. Section 2 describes the CyberShake applications, its scientific aspects, and its computational challenges and requirements. Section 3 gives an overview of the systems used to manage CyberShake workflows. Sections 4–6 describe the scalability challenges in the workflow description, management, and performance analysis and the solutions we developed to meet them. Section 7 shows the results of a number of CyberShake runs and the types of performance analysis performed on them. Section 8 gives an overview of related work. Section 9 concludes the paper.

2. CyberShake

2.1. Science description

Seismologists quantify the earthquake hazard for a location or region using the PSHA technique by estimating the probability that the ground motions at that site will exceed some intensity measure (IM) over a given time period. These intensity measures include peak ground acceleration, peak ground velocity, and spectral acceleration. PSHA seismic hazard estimates are expressed in statements such as: a specific site (e.g. the USC University Park Campus) has a 10% chance of experiencing 0.5 g acceleration or greater in the next 50 years. This type of ground motion estimate is useful for civic planners and building engineers because it provides a probabilistic estimate of the peak ground motions at a site over a specific period of time in the future. Although actual future peak ground motions are not known, very low probability peak ground motions may be ignored.

PSHA estimates are delivered as PSHA hazard curves (see Fig. 1) that present ground motion values (e.g. accelerations in g) on the x-axis, and the probability of exceeding these ground motion level within one year on the y-axis.

A set of hazard curves, for many sites in a geographical region, may be integrated into a regional hazard map (see Fig. 2) by keeping either the IM or probability constant and plotting the other as a function of geographic location. PSHA calculations require two essential inputs. First, a list of all possible future earthquakes that might affect a site is needed. (This information is available from an Earthquake Rupture Forecast (ERF).) Secondly, a way of calculating the ground motions that will be produced by each earthquake in the list is needed. Traditionally, the ground motions produced by each earthquake are calculated using empirical attenuation-based relationships. However, this approach has limitations. The data used to develop the attenuation relationships do not cover the full range of possible earthquake magnitudes. Also, attenuation relationships do not include basin or rupture directivity effects.

To address these issues, CyberShake uses physics-based 3D ground motion simulations with anelastic wave propagation models to calculate the ground motions that would be produced by each of the earthquakes in the ERF. Scientists involved in CyberShake believe that this new approach to PSHA has clear scientific advantages and can improve the accuracy of the seismic hazard information currently available. For a typical site in Los Angeles, the latest ERF available from the USGS (UCERF 2.0) identifies more than 7000 earthquake ruptures with magnitude > 5.5 that might affect the site. For each rupture, we must capture the possible variability in the earthquake rupture process. So we create a variety of hypocenter, slip distribution, and rise times for each rupture to produce over 415,000 rupture variations, each representing a potential

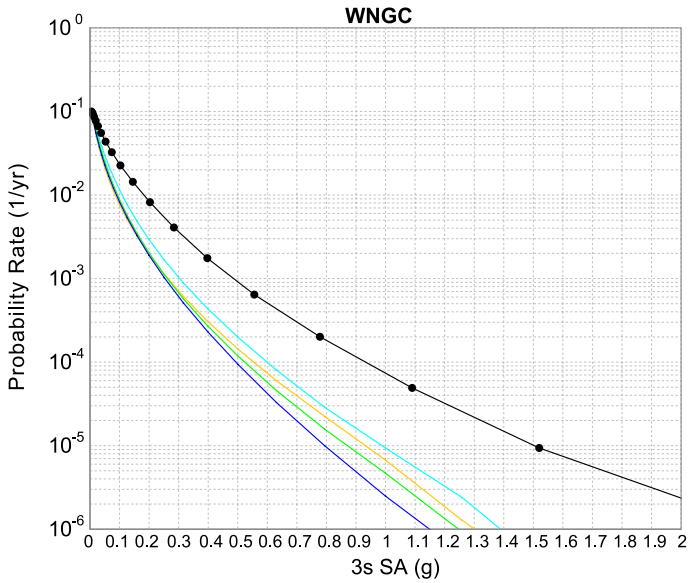

Fig. 1. Five hazard curves for the Whittier Narrows site near Los Angeles. The black line is the physics-based CyberShake simulation results. The colored lines are common attenuation relationships. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

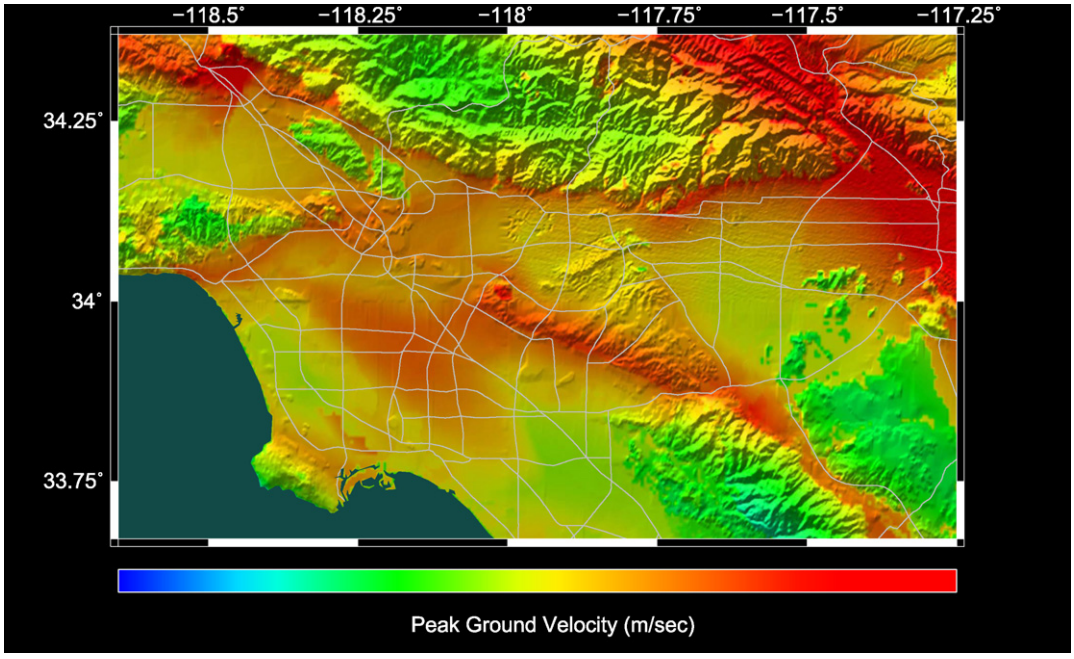

Fig. 2. An example hazard map for the Southern California area generated using attenuation relationships. Colors show the peak ground velocities that have a 2% chance in 50 years of being exceeded. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

earthquake. In CyberShake processing, there is a fairly technical, but important, distinction between ruptures (~ 7000) and rupture variations ($\sim 415{,}000$). These distinctions impact our workflows. The SGTs calculations generate data for each rupture, but post-processing must be done for each rupture variation.

Once we define the ruptures and their variations, CyberShake uses an anelastic wave propagation simulation to calculate strain Green tensors (SGTs) around the site of interest. Seismic reciprocity is used to post-process the SGTs and obtain synthetic seismograms [19]. These seismograms are then processed to obtain peak spectral acceleration values, which are combined into a hazard curve (Fig. 1). Fig. 3 contains a workflow illustrating these steps. Ultimately, CyberShake hazard curves from hundreds of sites will be interpolated to construct a physics-based seismic hazard map for Southern California.

In 2005, the CyberShake curves were calculated using a smaller number of ruptures and a smaller number of rupture variations, only about 100,000 rupture variations per site [10]. Since then, a new ERF (UCERF 2.0) was released by the

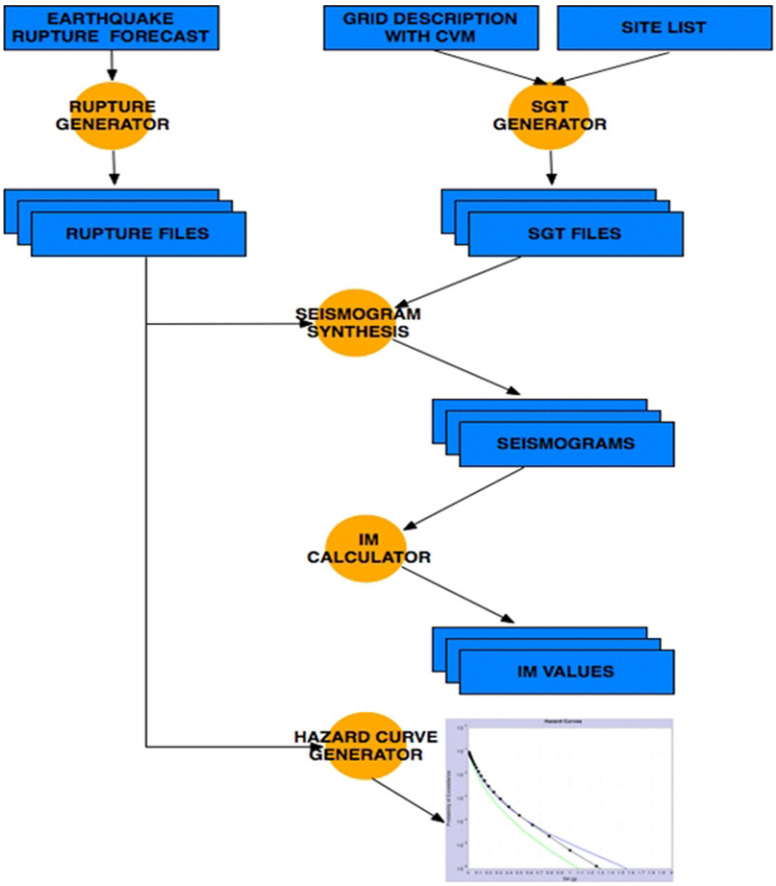

Fig. 3. A high-level CyberShake workflow.

Table 1
Data and CPU requirements for the CyberShake components, per site of interest.

Component	Data	CPU hours
Mesh generation	15 GB	150
SGT simulation	40 GB	10,000
SGT extraction	1 GB	250
Seismogram synthesis	10 GB	6,000
PSA calculation	90 MB	100
Totals	66 GB	17,000

Working Group on California Earthquake Probabilities [20]. This new ERF identifies more possible future ruptures than were used in 2005. In addition, based on initial CyberShake results, SCEC scientists decided to specify more variability in the rupture processes. As a result, the number of rupture variations we must manage increased by more than a factor of 4.

2.2. Computing requirements

The CyberShake platform must run many jobs and manage many data files. Thus, it requires extensive computational resources. An outline of computational and data requirements is given in Table 1. To compute the SGTs, a mesh of about 1.5 billion points must be constructed and populated with seismic wave velocity information. The velocity mesh is then used in a wave propagation simulation for 20,000 time steps.

Once the SGTs are calculated, the post-processing is performed. Processing is done for each of the rupture variations Post-processing begins by selecting a rupture variation. Then, SGTs corresponding to the location of the rupture are extracted from the volume and used to generate synthetic seismograms, which represent the ground motions that the rupture variation would produce at the site we are studying. Next, the seismograms are processed to obtain the IM of interest, which, in our current study, is peak spectral acceleration at 3.0 seconds. Each execution of these post processing steps takes no more than a few minutes, but SGT extraction must be performed for each rupture, and seismogram synthesis and peak spectral acceleration processing must be performed for each rupture variation. On average, 7000 ruptures and 415,000

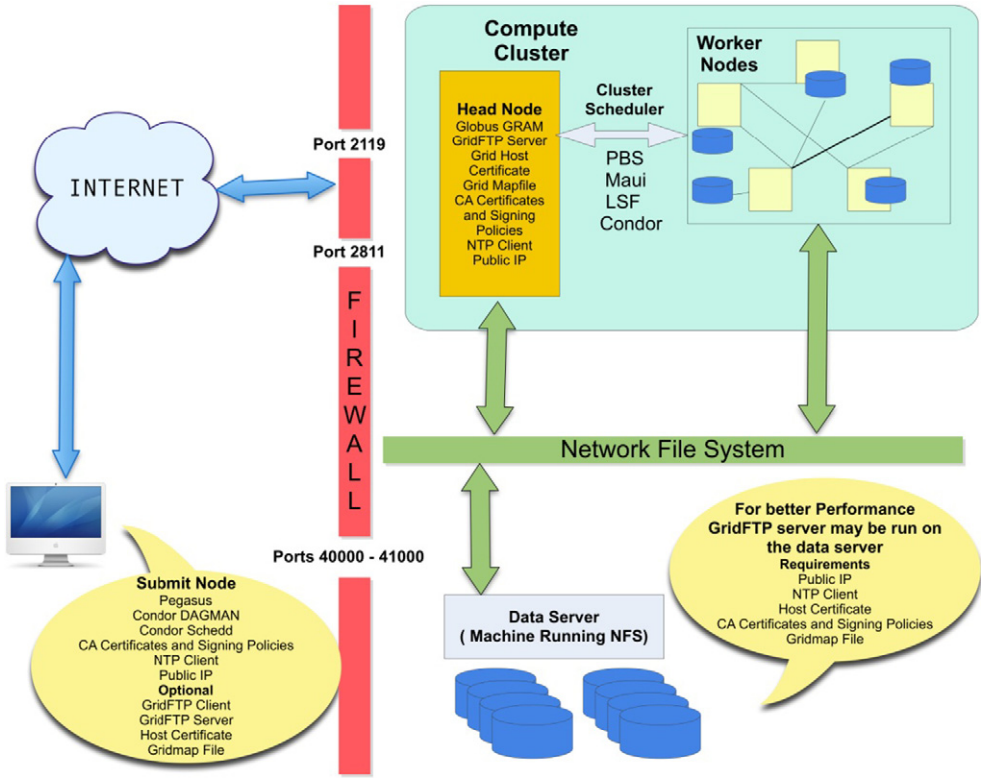

Fig. 4. Execution environment.

rupture variations must be considered for each site. Therefore, each site requires approximately 840,000 executions, 17,000 CPU-hours and generates about 66 GB of data.

Considering only the computational time, performing these calculations on a single processor would take almost 2 years. In addition, the large number of independent post-processing jobs necessitates a high degree of automation to help submit jobs, manage data, and provide error recovery capabilities should jobs fail. Such capabilities are provided by workflow management systems such as Pegasus [21–24]. The velocity mesh creation and SGT simulation are large MPI jobs that run on a cluster using spatial decomposition to distribute work to multiple processors. The post-processing jobs have a very different character, as they are "embarrassingly parallel" – no communication is required between jobs.

These processing requirements indicated that the CyberShake computational platform requires both high-performance computing (for the SGT calculations) and high-throughput computing (for the post-processing). To make a Southern California hazard map practical, time-to-solution per site needs to be short, on the order of 24–48 hours. This emphasis on reducing time-to-solution, rather than categorizing the system as a capability or capacity platform, pushes the CyberShake computational platform into the high productivity computing category, which is emerging as the key capability needed by science groups. In contrast to capability computing (a single, large job) and capacity computing (smaller, multiple jobs, often in preparation for a capability run), high productivity computing focuses on high throughput jobs with extremely short runtimes. The challenge is to minimize overhead and increase throughput to reduce end-to-end wall-clock time.

Given the above requirements for a single workflow and the number of workflows that need to be run, the NSF-funded TeraGrid [16] has been targeted as the execution environment for CyberShake, and based on our past experienced, we used Pegasus as our workflow management system.

3. Systems used to manage CyberShake workflows

3.1. Execution environment

In general, we can view the execution environment as being composed of a *submit node* or *submit host,* where the workflow management and log analysis system reside, and the broader, potentially remote cyberinfrastructure. Fig. 4 shows a typical deployment of the Pegasus Workflow Management System that manages the execution of data management and computational tasks on remote resources. The figure shows the software that needs to be installed on the submit node for remote management and authentication to remote resources that span administrative domains. The figure also differentiates the remote cyberinfrastructure by identifying a *head node* that is responsible for receiving remote job submissions and for

scheduling them onto *worker nodes* that are local to it. In most cases, the worker nodes share a common file system and thus conduct communications, such as data file exchanges, through this common infrastructure. Good shared file system performance is thus important for data-intensive applications. Fig. 4 also shows data storage resources, where data needed for the workflow and generated by the workflow can be stored and accessed by both the worker nodes and the submit node. To enable remote communications, firewall ports need to be opened on the cyberinfrastructure.

3.2. Workflow management system

The Pegasus Workflow Management System (or Pegasus) is composed of the Pegasus mapper, the DAGMan [25] workflow execution engine, and the Condor [28] task manager. The mapper can map workflows onto a variety of target resources such as those managed by PBS [26], LSF [27], Condor, and individual machines. The executable workflow produced by the mapper has directives to DAGMan for the execution of the workflow components. These directives include remote job execution, data movement, and data registration. Authentication to remote resources is done via GSI [29]. Mapping the workflow to an executable form involves finding resources that are available and can perform the computations as well as locating the data used in the workflow and the necessary software. We assume that data may be replicated in the environment and that users publish their data products into some data registry. This registry can be a private or a community resource.

Pegasus uses the logical filenames referenced in the workflow to query a data registry service such as the Globus Replica Location Service (RLS) [30] to locate the replicas of the required data. Given the set of logical filenames, RLS returns a corresponding set of physical file locations. Optionally, Pegasus also adds nodes to the workflow to register the final and intermediate workflow data products into the registry. In this way, new data products can be easily discovered by the user, the community, or another workflow. In order to discover the locations of the logical application component names (transformations) defined in the workflow instance, Pegasus queries the Transformation Catalog (TC) [31] and obtains the physical locations of the transformations (possibly on several systems) and the environment variables and libraries necessary for the proper execution of the software. Pegasus also supports staging of statically linked executables on demand. In that case, the executables are treated as input data for the corresponding workflow tasks. The executables are transferred to the remote grid sites along with other input data required by the jobs.

The executable workflow produced by the mapper is given to DAGMan for execution. DAGMan manages the job execution by determining when jobs are ready to run, submitting jobs to Condor and Condor-G. The latter sends jobs to remote resources via Globus. Condor can throttle the number of jobs released so as to not overwhelm either the submission host (where the jobs originate) or the remote platform (where the jobs execute). If a job fails, DAGMan will automatically retry the failed job and create a rescue DAG if the job cannot be completed successfully. The rescue DAG contains the portion of the original workflow that has not been executed. This restart capability is critical for CyberShake. Jobs will fail for a variety of reasons, and being able to resume execution easily is a major benefit.

During the workflow execution, Pegasus captures provenance information about the execution tasks. Provenance includes a variety of information, such as the hosts where the tasks have executed, the runtime, environment variables, etc.

These tools enable CyberShake to be run on a variety of platforms and be managed from a single local job submission host.

3.3. Workflow performance analysis

Pegasus also wraps the computational workflow tasks with the kickstart wrapper [32], which collects performance statistics and provenance information about the execution of the tasks and allows us to check the return codes for successful execution. Kickstart records are easy to mine for usage statistics using the NetLogger Toolkit [33], a methodology and a set of tools for troubleshooting and performance analysis of distributed systems. NetLogger automates the data analysis by setting up a "pipeline" that can apply streaming analysis functions while loading all the logs from one or more runs into a single set of database tables. NetLogger extracts and analyzes this data using a combination of SQL stored procedures and functions written in the statistical analysis language, R [34]. Fig. 5 shows the interaction of the NetLogger log pipeline with the Pegasus-WMS.

The parsing/streaming analysis stage of the pipeline is performed by a modular framework that reads logs in various formats and outputs NetLogger's own "Best Practices" (BP) log format [35]. The tools are flexible enough to easily process the CyberShake logs, which are organized into a directory tree containing thousands of files, with a single invocation. Configuration files map each type of log to a log processing module. These log processing modules are programs that can emit zero or more records for every record consumed. Thus, they can condense, combine, or otherwise transform the data. These transformations are currently limited by the restriction that only one log file is seen by each module, and more fundamentally by the potential performance impact of holding onto large amounts of internal state. However, within these restrictions, a number of useful transformations can occur. Not surprisingly, the most common transformation is to reduce the amount of data by simply eliminating uninformative fields; another common transformation is to calculate the duration between two timestamped log events that can be linked using an identifier.

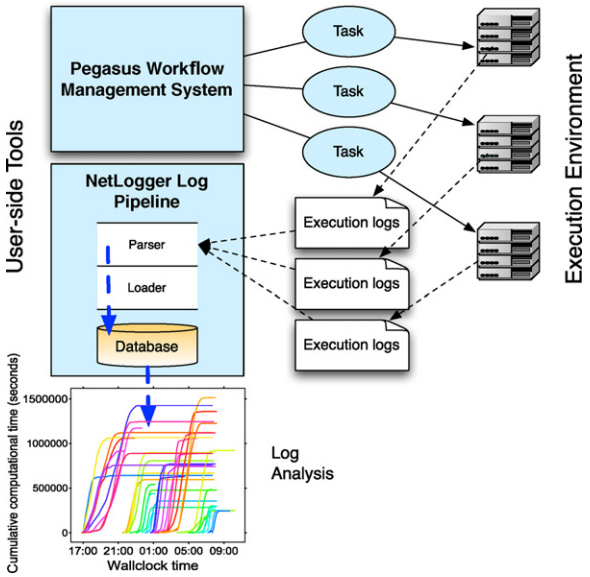

Fig. 5. A view of the integrated Pegasus-WMS and NetLogger.

4. Describing CyberShake workflows in a scalable manner

Over the past few years, CyberShake has been using the Pegasus Workflow Management System [22–24] to support the scalable and reliable execution of the workflow-based computations. With time, the scale of the analysis needed for a single geographic site has risen from approximately 250,000 to ∼840,000 individual tasks.

There are many workflow description languages [1,2] including graphical [6,7,36,37] and scripting [38,39] languages. In our work, the workflow is expressed in a simple XML format, the DAX (directed acyclic graph in XML), which describes the logical workflow components, their input and output files (in logical terms), and the dependencies among the components [40]. This XML description is execution platform independent and can be thought of as an intermediate workflow language that Pegasus uses as input. However, most applications, including CyberShake, use a *workflow generator* written in a higher-level language, such as Java or python, to write out the XML. For quarter of a million tasks, the workflow generator can write an XML file that expresses the workflow at a single level of nesting (no recursive workflow definitions). However, as the size of the workflow grew, so did the size of the XML representation. The XML representation for the 840,000 task CyberShake workflow consumed 1 GB. As a result, Pegasus needed an ever increasing amount of memory to parse larger DAXes. In the end, the non-nested workflow solution did not scale. Thus, although we wanted to keep the intermediate representation as simple as possible and leverage the higher-level language constructs to represent workflow complexity, we needed to add the concept of nesting or recursion to the intermediate workflow representation to obtain scalability. As a result, the workflow can be specified in a multi-hierarchical way. The recursion is not only at the level of planning, but also at the level of execution. Thus, each sub-workflow is planned and executed before the dependent sub-workflows are planned. Fig. 6 shows the approach in more detail. In this example, the top level workflow is composed of four tasks, where task A3 is a sub-workflow, which is in turn composed of two other sub-workflows B3 and B4. Each sub-workflow has an explicit workflow planning phase and a workflow execution phase. Fig. 7 illustrates the sequence of workflow planning and execution for the workflow in Fig. 6. The execution of the top level workflow and the execution of the sub-workflows is done by DAGMan. We also see that we can have multiple task mapping activities while other application tasks in the workflow are executing. Because each sub-workflow is planned just-in-time, the size of the sub-workflows can regulate how close to the sub-workflow execution resource selection is done. Thus, for a very dynamic execution environment, the size of the sub-workflows should be relatively small.

For the CyberShake workflows, only a single level of recursion is used. Since the CyberShake workflow can be viewed as any number of independent sub-workflows, a number of different variations can be tried. However, there are system limitations and other considerations that guide some of the workflow design decisions. For example, having all 80 independent sub-workflows (each with 10,000 tasks) running simultaneously can put an excessive load on the submission host that handles the workflow and task execution. Too many concurrent jobs can also overwhelm the cyberinfrastructure resources, such as remote job submission systems and data transfer servers. One solution is to introduce artificial dependencies into the outer-level workflow, for example by creating “pipelines” or chains of sub-workflows with parent–child relationships so that the chain elements execute serially.

Recursive workflows can also improve the use of the file system on the remote resource. Since each sub-workflow corresponds to a directory on the remote host that contains all the files needed for execution of the sub-workflow, running multiple sub-workflows reduces the number of files per directory, improving file system response.

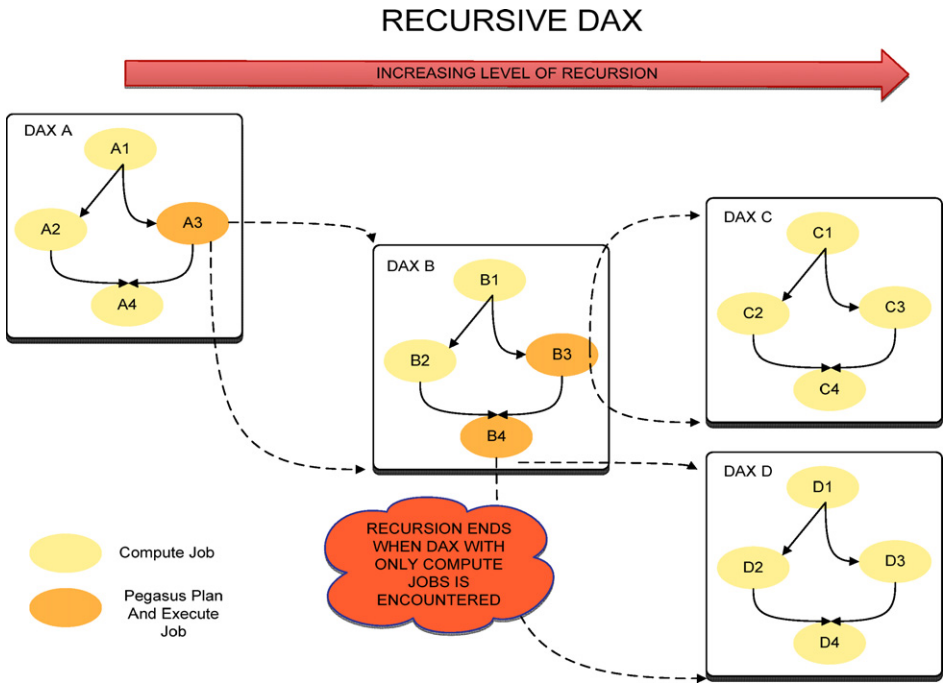

Fig. 6. An example recursive workflow. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

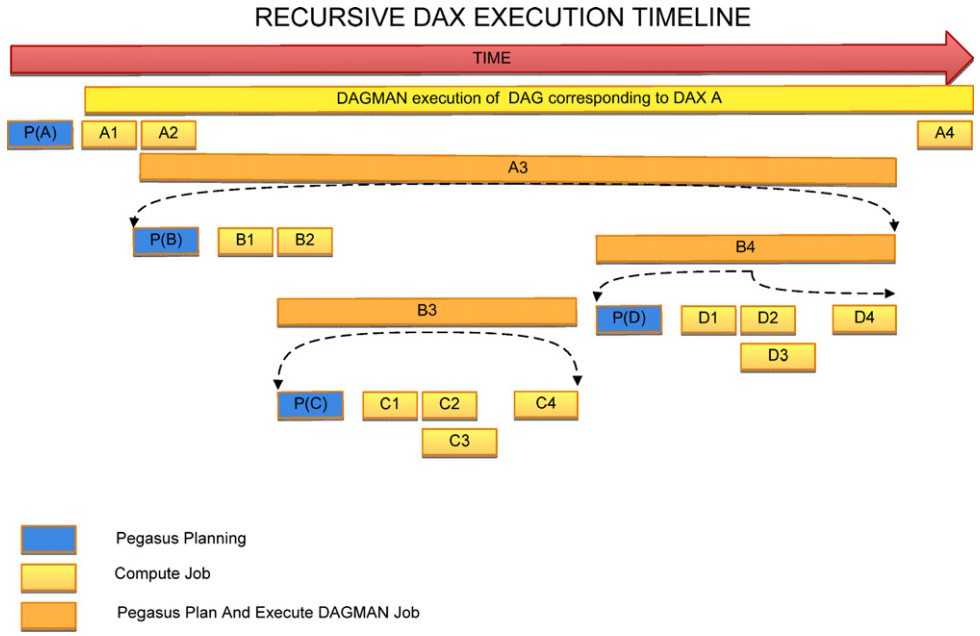

Fig. 7. A time line showing the sequence of task planning and execution for the workflow in Fig. 6. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

Following the SCEC example, we can have 10 pipelines, each with 8 sub-workflows, so that 10 sub-workflows run simultaneously. When one of the 10 initial workflows finishes, its child sub-workflow can be planned and executed, and so on until all 10 pipelines complete. However, if a sub-workflow fails, that entire pipeline would stop execution, since jobs further down the pipeline cannot run until their parent finishes execution. Thus, the artificial dependencies introduced to throttle the execution of the workflows can have an inadvertent impact on the overall workflow execution. The dependencies prevent the execution of tasks that would under other circumstance be able to proceed with execution.

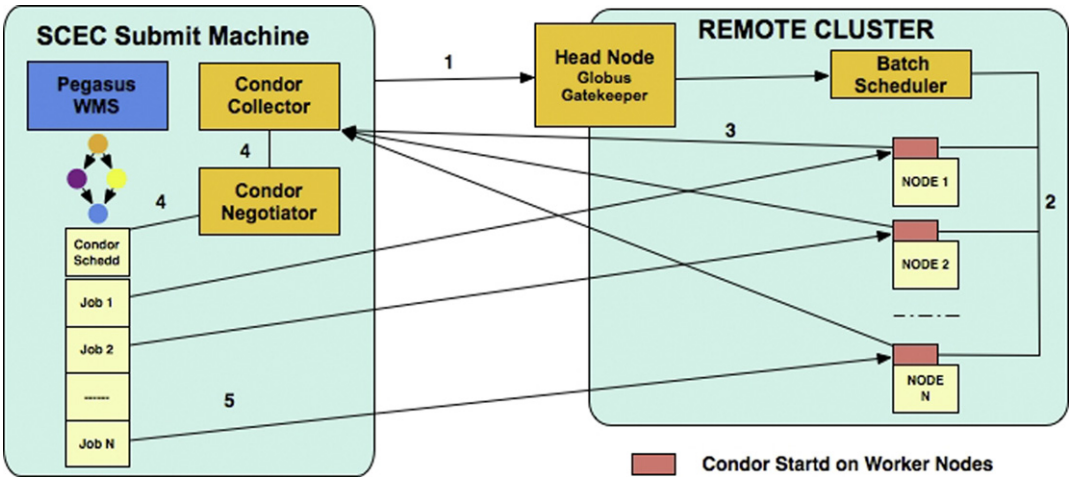

Fig. 8. Steps involved in acquiring glideins. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

5. Achieving scalable workflow mapping and execution

Since the SGT simulation requires large MPI calculations, while the post-processing involves high throughput, embarrassingly parallel jobs, we decided to separate these stages into two independent workflows. This gives us flexibility to run the two pieces at different times on different platforms, since some environments may be optimized for MPI jobs, while others are more efficient for short serial jobs.

5.1. Finding the right execution site

Initially, we targeted our CyberShake workflows at the NCSA's Abe cluster for both SGT and post-processing workflows. However, we ran into complications. We attempted to verify the SGTs by using them to generate a seismogram for a certain source and comparing it against a seismogram generated for the same source by a previous simulation. However, we were unable to successfully verify the SGTs and despite investigation were unable to pinpoint the source of the error. The post-processing workflow also presented unexpected challenges. Sometimes seismogram synthesis jobs would experience a segmentation fault that would not reoccur if the job was rerun. After extensive memory profiling of the seismogram synthesis code (written in C) and examination of the runtime environment, we concluded that the version of the Linux kernel used by Abe, 2.6.9, may contain a memory management bug. On multi-core systems, memory claimed by the cache is not made properly available to running code, and therefore segmentation faults can occur even though there is sufficient memory available. This problem was exacerbated by the short runtimes (< 1 min) and high memory requirements (1.6 GB) of the synthesis jobs. As a result of these issues, we postponed running CyberShake on Abe until the kernel is upgraded.

We switched our target platforms to NCSA's Mercury cluster for the post-processing workflow and USC's campus cluster for the SGT generation. Separating the two stages onto different platforms was trivial; we only had to transfer the two SGT files that are the result of the first workflow to Mercury and register them in the Globus RLS. This also permitted us to utilize computational resources at two remote sites. This workflow execution demonstrates the value of the Pegasus workflow approach, which defines abstract workflows. Modifying the workflow to run on a different physical system is simply a matter of changing configuration information and replanning the workflow.

5.2. Provisioning resources ahead of the workflow execution

Many remote execution environments limit the number of jobs a single user can put in their queue. Additionally, remote cluster schedulers often have a scheduling cycle of several minutes, meaning that when jobs finish, it can take some time for another job to be scheduled. To address these concerns, we use Condor glideins [41], a type of multi-level scheduling [42]. Fig. 8 shows the steps involved in using glideins to create a temporary Condor pool: (1) The user requests a group of nodes for a specified duration via Condor. Condor sends the request to the Globus gatekeeper on the remote host, which submits the request on to the batch scheduler. (2) After waiting in the remote queue, the glidein job begins on the requested nodes. (3) The nodes start up the Condor startd process and advertise themselves back to the Condor collector on the local job submission host as available nodes. This creates a temporary Condor pool on the remote platform. (4) The Condor Negotiator queries the Condor Collector for the state of resources in the pool and queries each condor_schedd that has waiting resource requests in its job queue. The negotiator tries to match available resources with those requests. (5) The local submission host can then schedule directly on the remote nodes by matching queued jobs to available resources and sending the job to the node's startd process, which then begins the job.

5.3. Task clustering

An immediate problem was how to execute over 840,000 jobs without overwhelming the scheduler on either the submit host or on the execution platform. The average runtime for SGT extraction jobs ("extraction jobs") is 139 seconds; for seismogram synthesis jobs ("synthesis jobs"), 48 seconds; and for peak spectral acceleration jobs ("PSA jobs"), 1 second. Running these short runtime jobs individually through the scheduler would mean that computing resources would sit idle much of the time, waiting for the scheduler to schedule jobs, and the scheduler would be under intense load to schedule new jobs to waiting idle processors.

The glide-ins reduce the load on the remote execution site. Because the remote scheduler (on Mercury, PBS) sees only a single job (the glide-in), minimal load is added. However, the scheduler on the submit host can still be overwhelmed with a large number of tasks, and the overheads in sending a large number of short duration tasks over the network can be significant. To alleviate these problems, we use a Pegasus technique called "clustering," in which jobs of the same type are grouped together [43]. Pegasus automatically restructures the DAX so that a number of individual jobs are replaced by a single call to *seqexec* (a Pegasus remote-site executable), which iterates over a Pegasus-created file containing a list of jobs and sequentially executes them on a remote resource. Each DAX, and therefore each DAG, contains fewer jobs, reducing the load on the local Condor scheduler.

Selecting the right clustering parameters for a given workflow and a given execution platform can be difficult. Initially, we selected clustering values of 20 for the extraction jobs, 40 for seismogram synthesis jobs, and 200 for peak spectral acceleration jobs.

Monitoring our execution, we discovered we had poor utilization; many available processors spent long periods of time idle. This was due to high clustering values. Each DAX has only about 20 extraction jobs, but each extraction job has between 2 and 1568 synthesis child jobs. In order to keep processors busy, the extraction jobs must be completed quickly so that the more numerous synthesis jobs can begin execution and claim idle processors. When clustering is used, dependencies are moved to the cluster level. Instead of a synthesis job having an extraction job parent, a synthesis cluster has an extraction cluster parent. Using a cluster factor of 20, it took at least 45 minutes before any synthesis clusters would start execution. We found that for most of the duration of the workflow, not all the processors were used as a result.

The clustering parameters were adjusted to 2 for extraction jobs, 10 for synthesis jobs, and 60 for spectral acceleration jobs. With these new parameters, extraction clusters completed in a few minutes, allowing synthesis clusters to start more quickly and run on previously idle processors. Although this increased the total number of clusters fourfold, it reduced average workflow completion times by a factor of three, since the processors spent much less time idle. We also doubled the number of sub-workflows, from 40 to 80, so that the number of files per directory on the remote host was kept to a reasonable level (around 30,000).

5.4. Modifying parameters of task submission

As we mentioned in Section 4, achieving scalability in the workflow representation requires the use of recursive workflow definitions and may result in too many parallel tasks being managed by the local scheduler. Because adding artificial dependencies also did not solve the problem, we altered the workflow to contain a pool of 80 independent sub-workflows rather than pipelines and used the Condor parameter *maxjobs* to limit the number of concurrent sub-workflows. If one sub-workflow failed, it did not hold up others; instead, a new sub-workflow was pulled from the pool, planned, and run. Changing to a pool approach increased the amount of automation and decreased runtime for the workflow.

5.5. Dealing with cyberinfrastructure failures

We noticed that the most common cause of workflow (or sub-workflow) failure was an error with the GridFTP stageout job at the end of the workflow. These transfer jobs were inserted automatically by Pegasus to transfer data products back to SCEC servers after completion of each workflow. If the transfer failed, the workflow was configured to retry it three times before failing and generating a rescue DAG, which registered as a failure in the outermost workflow. The outermost workflow would respond by starting the DAX over from the planning stage, and since for reasons of space we did not use the Pegasus checkpoint feature, all of the successful computations were thrown out and rerun. To avoid redoing completed computations, the outer-level workflow was altered to tap into the auto-restart feature of Condor 7.1, which checks to see if a rescue DAG exists and, if so, restarts from it. This prevented us from duplicating computations that had already been successfully executed.

Further investigation of the transfer failures revealed that the main factor was the large number of files being transferred. Each sub-workflow had two transfer jobs of approximately 5000 files, one job for seismograms and one for peak spectral acceleration files. Additionally, the files are quite small; the seismogram files are 24 KB each, and the spectral acceleration files are just 216 bytes. Using GridFTP to transfer so many small files was not efficient. We added two additional jobs to each sub-workflow which would zip all of each kind of file before transfer, reducing the number of files to transfer from over 800,000 to 160, two per sub-workflow. Additionally, transferring larger files lets us utilize the speedup advantage of GridFTP. Even with the additional time required for zipping, these improvements sped up the stageout of data products by a factor of ten. This helped us improve our processor utilization, as sub-workflows could now quickly perform their transfers, finish,

and clear the way for a new sub-workflow to start execution and claim any available resources. Modifying the workflow also had the side effect of improving our local processing. After stageout, we insert the peak spectral acceleration values into a local database. We then use the database to generate a hazard curve, the final result. Performing the database insertions from a zip file is much faster than from a large number of small files, by a factor of four in our case.

6. Scaling up workflow performance analysis

Given the number of workflow tasks and the proportionately large amount of logging information, efficiently parsing this information is critical. In our log processing, the BP log file output from the parsing stage is efficiently loaded into a relational database. This occurs as the data is produced, with minimal latency (roughly 1 second) between the time a record appears in a log file and the time it enters the database. Much earlier versions of the NetLogger Toolkit used a MySQL extension (with equivalents in most RDBMS products) to load data directly into tables from a text file. This is by far the fastest approach, but it requires that the application apply all table constraints and also lock the tables for the duration of the load. Our current approach, instead, uses a multiple-row INSERT operation, which although slower, keeps the tables unlocked and also allows database constraints to apply. An additional optimization for post-hoc data loading is to apply the table indexes after all the data files have been loaded. Using this optimization, the current implementation can load roughly 25,000 records per minute, or 37 M per day (into MySQL ISAM tables); with indexing and database uniqueness constraints, it achieves about half that rate. This limit is considerably higher than the current CyberShake output rates of around 1–2 M records per day, and even the current goal of a 4x speedup.

The set of database tables is generic; specific tables for CyberShake or Pegasus-WMS are derived as part of the analysis. In general, the analysis is performed using R code with the following simple structure: (1) Connect to the database; (2) Invoke SQL statements and/or stored procedures to fetch a database; and (3) Analyze and graph the dataset.

These analysis routines are packaged into R function libraries that can be easily loaded and used inside an interactive R session. For interactions with other programs, parameterized R programs can be invoked either as command-line programs or through the language's R binding.

Using this approach, complex correlations can be performed almost in real time. For example, to determine the proportion of time that each job spends waiting to run, running, waiting to terminate, and performing post-run actions, the information from the Condor DAGMan logs need to be correlated with the runtimes in the Kickstart execution logs. We currently implement this with a stored procedure that can be called from R to generate a table with a row for each job. The results can then be graphed as a Gantt chart showing the time breakdown in detail for a given sub-workflow; a chart produced by this method is shown in Fig. 14 in Section 7. For the CyberShake workflows presented here, the query for one Gantt chart – which summarizes roughly 10,000 execution times and correlates them with the several task clusters of 300–400 jobs – takes about 15 seconds (on a MacBook Pro; times will vary widely by hardware). This is sufficiently fast to make re-generating these charts on the fly feasible. Although performance is good for most queries, it is of course possible to come up with correlations that require much longer to perform. Our strategy in this case is to derive smaller, more specific tables with stored procedures that are run by built-in MySQL scheduling mechanisms.

R is used for both automated and interactive visual data exploration. For example, to produce the barchart of the execution and job times shown in Fig. 10, the full set of 800,000 timings (and job counts) was loaded into R; then the *aggregate()* function summed each by whole hours, and the *barchart()* function produced the graph. For the CyberShake data, the time computing in R is negligible, so variations on the graph can be quickly explored; the same dataset was used to generate all the execution time summaries, except the Gantt chart, shown in Section 7. Thus, although SQL alone can perform some simple summaries, for the most part SQL is used to load a memory-resident (correlated) subset of the data into R for more detailed analysis. We have found this combination of tools both scalable and useful for analyzing and exploring the CyberShake data.

7. Results

The computational results reported in this paper were the result of 9 CyberShake runs, occurring from June 26, 2008 to July 18, 2008. Table 2 shows the volume of tasks and the total end-to-end wall-time for each run. All calculations were performed using a reservation of 400 nodes (800 processors) on NCSA's IA-64 Mercury TeraGrid cluster. We used a single glidein job to claim all 800 processors. Since the maximum job length allowed in the main queue on Mercury is 25 hours, we resubmitted the glidein request daily to reclaim the processors. A detailed discussion of the science results can be found in Graves [44]. The results below were obtained using the NetLogger toolkit. In the remainder of this section, we will focus on the run for Site 1, which represents a single run of CyberShake to generate a hazard curve for the geographic location near the University of Southern California. In total, the USC run produced 417,886 seismogram files (24,000 bytes each) and an equal number of PSA files (216 bytes each), for a total of approximately 9.5 GB of data. The aggregate computation time and percentage of total time consumed by each job type for the USC run are shown in Table 3. The *Directory creation* jobs create a random directory for the workflow execution on a remote site. The *Transfer* jobs stage data out to the user-specified location. The *Registration* jobs register the newly created data products into a data registry. Clearly, the majority of time is spent in *Synthesis* jobs. This suggests examining both the workflow environment and the synthesis C code itself for optimization opportunities.

Table 2
Task volume and wallclock time for 9 CyberShake workflows from June 26–July 18, 2008.

Site	Pegasus jobs	Extraction	Synthesis	PSA	Zip	Total jobs	Failed jobs	Walltime (min)
1 (USC)	504	7000	417,886	417,886	80	843,356	264	1042
2	950	7093	418,256	418,256	154	844,709	483	1066
3	1074	7026	417,954	417,954	156	844,164	624	1287
4	896	6932	415,416	415,416	158	838,818	422	988
5	736	6912	415,778	415,778	156	839,360	268	1542
6	703	7045	426,740	426,740	156	861,384	235	964
7	946	6823	416,090	416,090	158	840,107	472	792
8	1552	6919	418,946	418,946	156	846,519	1084	1287
9	923	6947	417,772	417,772	156	843,570	455	890
All	8284	62,697	376,483	376,483	1330	7,601,987	4307	1095

Table 3
A summary of performance characteristics of tasks for the July 7–8, 2008 CyberShake workflow for the USC site.

Job type	Hours	% of total time
Directory creation	0.01	0.0001
Registration	0.06	0.0007
Transfer	15.6	0.2
Extraction	300.6	3.7
Synthesis	7494.6	93.5
PSA	175.2	2.2
Zip jobs	16.0	0.2
Total	8017.34	100.0

To see the overall progress of each of the 10 sub-workflows in the recursive DAX pipeline described in Section 4, we use R to subset and color the computations by their sub-workflow number. This plot, the left side of Fig. 9, shows the sub-workflow numbers lined up beneath the start of the corresponding sub-workflow to clearly illustrate the simultaneous start of the first 10 sub-workflows, followed by somewhat less synchronized but still visible starts of the next three sets of 10 sub-workflows. A closer examination of the figure shows that some of the sub-workflows consumed much less computational time than others. The reason for this is not because they ran fewer jobs but because the synthesis jobs in these sub-workflows took less time to complete. The runtime of a synthesis job is proportional to the number of points on the rupture surface for that rupture variation. The largest rupture in the workflow has 13,152 points, whereas the smallest has only 75. In general, larger magnitude ruptures have more points on their surfaces. Additionally, all the rupture variations from a single rupture are in the same sub-workflow. As a result, multiple rupture variations that have similar runtimes end up grouped together, further contributing to variance in runtime between sub-workflows. This can be seen by examining the time taken in the synthesis jobs broken down by sub-workflows and shown as parallel box plots in the right-hand panel of Fig. 9. It is clear that there is an almost perfect correspondence between the "short" curves in the panel on the left and the lower median runtime for synthesis jobs on the right. For example, the difference between the tallest curve, sub-workflow 29, and any of the small curves for sub-workflows 31 to 39 matches the roughly 4:1 ratio in medians (black dots) for these same sub-workflows shown in the right-hand graph. However, these differences are not seen at the workflow-level, where we see a steady progress of the jobs and computational time. Fig. 10 shows the number of jobs executed per hour and the cumulative amount of time the jobs required. We can see that the proportions of jobs completed and the cumulative runtime remain constant for the first 15 hours (with an average of 54,449 jobs per hour). In hours 16-18, towards the end of the workflow, data management tasks such as data registration and data transfer are occurring in the workflow, and thus we see many jobs with little or no computational time.

Although we see a balance at the application level, there may be some imbalances at the system level. They may indicate a failure in workflow planning or hardware problems. To examine this, we start with a non-parametric histogram, or "density plot", of the distribution of computational time by worker node as shown in Fig. 11. The "rug" plot at the bottom shows the actual data points; each line corresponds to the total number of hours spent on a given worker node. From this plot, it is clear that the computational time is normally distributed with no large groups of outliers. A more detailed view of the computational time broken down by job type and host is shown in Fig. 12. In this figure, a scatterplot is drawn for the different types of jobs (see Table 3). In each scatterplot, a point at the intersection of "host" X and "hours" Y represents Y total hours of computation for that host. The hosts are sorted by the total amount of computation done by that host, from least to greatest. Thus we see an S-shape for the Synthesis job, which dominates the overall time, corresponding to the normal distribution in Fig. 11. The rest of the scatterplots show random variation, indicating no particularly "bad" or "good" hosts for these computations. Visualizations such as this are useful for verifying that there are no strong patterns or groups of outliers for any given job type or host, something that is difficult to do reliably with statistical reductions of the data.

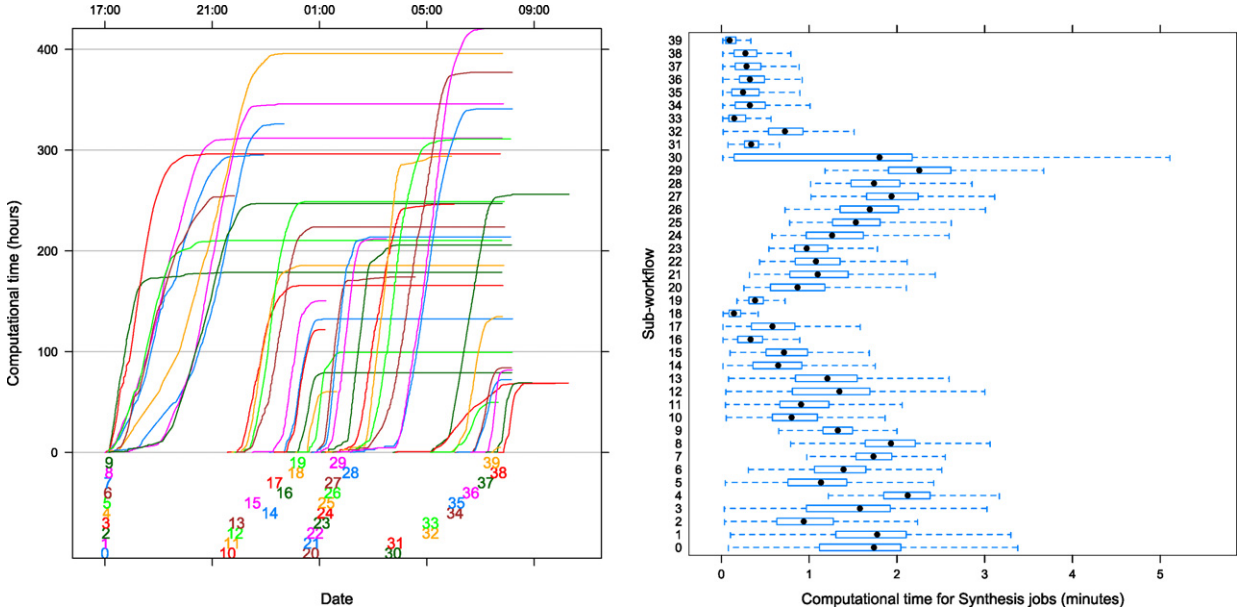

Fig. 9. (left) Progress of CyberShake sub-workflows through time, (right) performance of synthesis jobs in sub-workflows. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

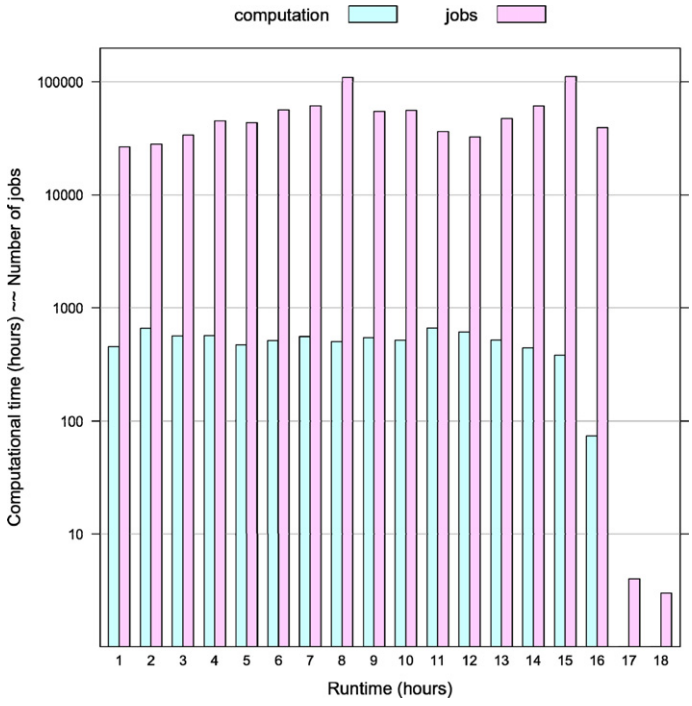

Fig. 10. Cumulative number of jobs and their cumulative runtime per hour of execution. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

So far, we examined the performance of the workflow in terms of the individual application tasks. However, as we mentioned earlier, the tasks were clustered for performance reasons: to reduce the loads on the task submission system and scheduling system and also to reduce the wide area network delays. Fig. 13 shows the number of different clusters in the workflow for each job type. As we can see, the largest fraction of clusters is formed by seismogram jobs.

Examining the performance of the workflow at the cluster level allows us also to reason about the overheads contributed by the workflow management system.

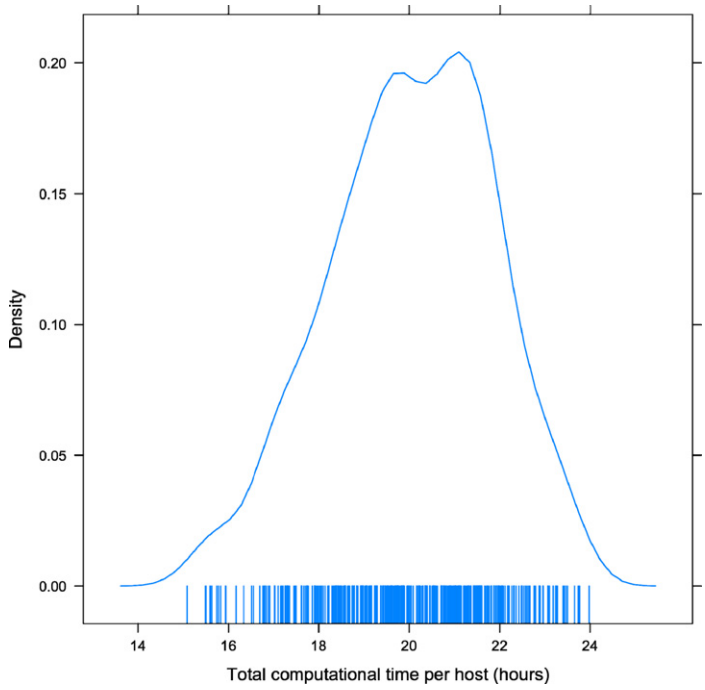

Fig. 11. Density plot of the distribution of jobs across computational hosts.

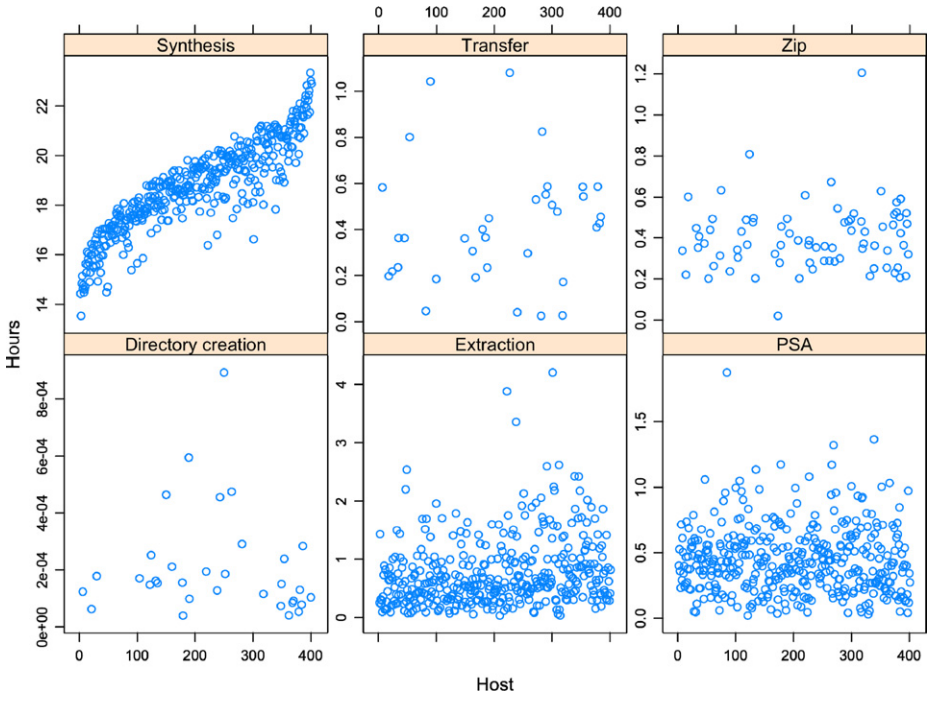

Fig. 12. Sum of computational time for each worker host, divided by job type. Worker hosts (on the *x*-axis) are ordered by increasing total computational, thus the pattern in Synthesis, which accounts for the bulk of that time.

Fig. 14 shows the execution progress of clusters from sub-workflow 29. It breaks down the execution time of individual clusters between the actual cluster execution and the Condor overheads. The overheads include the amount of time a cluster waits in the Condor queue for execution, which is dependent on the number of available resources and network connectivity. The overheards also include the time to stage out standard error and output files for the cluster. These transfer times are

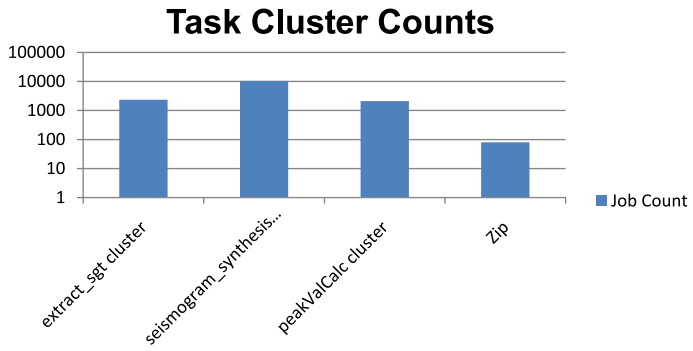

Fig. 13. Total number of clusters in the USC CyberShake workflow. Note that the *y*-axis is log scale.

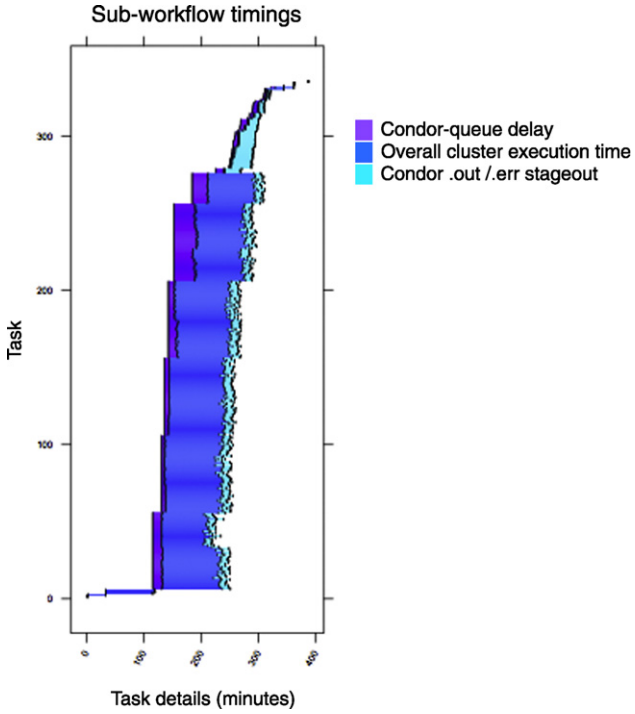

Fig. 14. Progress of task clusters within a sub-workflow. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

dependent on the sizes of the files and on network connectivity. We see that towards the end of the partition, the data transfer overheads were non-negligible.

Finally, Fig. 15 shows the relative contributions of the actual application runtime and the overheads to the total execution time of a cluster. The figure shows the averages over all the clusters in the workflow. The *Condor delay* is the waiting time of the cluster in the Condor queue. The *DAGMan delay* measures the time between the point where a job in the workflow is ready to execute (its dependencies are satisfied) and the time that DAGMan releases it for execution. The *Cluster Code delay* is the overhead of managing the execution of a cluster on a remote resource. As we can see, the Cluster Code delay is particularly noticeable for the short running PeakValCal jobs. These were placed into clusters of 200 individual jobs (compared to 40 seismogram synthesis jobs per cluster). The overhead shown in Fig. 15 cannot be simply explained as the Condor stage out delay (see Fig. 14). We investigated the issue and discovered that a feature in the cluster execution code was causing a slowdown. By default, the code would report the progress of the jobs in the cluster to a global log file and/or to a individual file on the submission site. This reporting enables the user to easily check how the execution of a cluster is progressing. It is especially important in the case where the number of constituent jobs in a cluster is high, and each of the individual jobs take some time to run. However, writing to a global log did not scale, especially since the tasks within the PeakValCalc clusters were of very short duration. For subsequent CyberShake executions, the global logging feature was turned off.

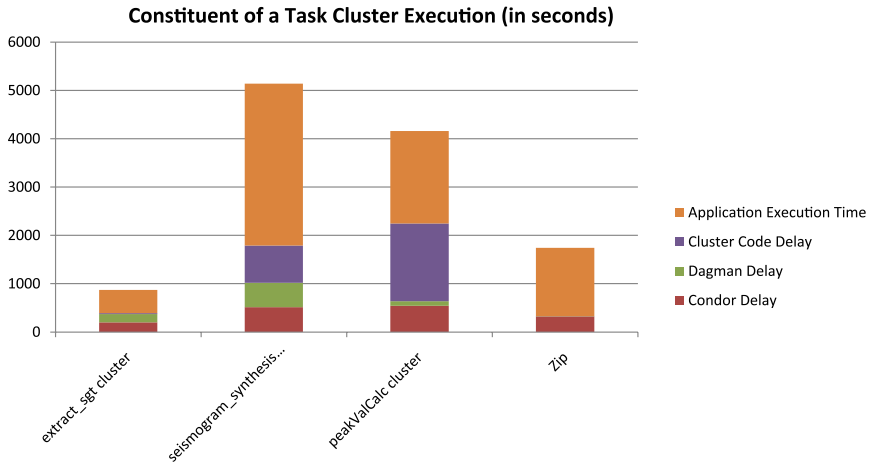

Fig. 15. Average system delays in the execution of clusters and the average application execution time within the clusters. (For interpretation of the colors in this figure, the reader is referred to the web version of this article.)

8. Related work

There are a number of research areas [1,2,45] that relate to the work presented here.

8.1. Workflow-based applications

As mentioned in the introduction and described in a number of publications [1,2,46–49], many scientific applications today make use of workflow technologies to achieve science goals. However, many of these applications have a relatively small number of computational steps. In our work, the CyberShake workflows are large-scale with hundreds of thousands of individual tasks, and because of their scale good performance is critical.

8.2. Workflow types and workflow systems

Workflow systems can be broadly divided into ones that support the composition of standalone applications and those that support the composition of services. Much of the work in industry has focused on the definition of standard workflow languages such as BPEL [38] and on the development of workflow engines capable of executing BPEL-specified workflows [50,51]. Some science disciplines, especially bioinformatics, are using BPEL and BPEL-like workflow specifications to orchestrate the services that are available in their communities [7,39,52].

Expressing applications as services is not applicable in many scientific disciplines, where performance and scalability are critical and where the service invocation overheads are simply too expensive. Thus, the workflows are often expressed as workflows of standalone applications. There are many workflow systems that support application component composition and execution. Some, such as Triana [53], Kepler [36], and VisTrails [54], provide graphical user interfaces for workflow composition, while systems such as Karajan [55] provide a scripting language to specify the workflow. Some of these workflow system support more complex control structure loops, conditionals, and hierarchical workflow definitions. However, in our work we found that these can be implemented at the workflow generation level (for example, using Java or python) rather than in our intermediate Pegasus description. Additionally, our hierarchical workflow definition relates not only to the workflow structure (as in the other systems) but also to the order and timing of mapping the portions of the workflow to the computational resources.

8.3. Workflow optimization

Today, few workflow systems perform workflow-level optimization and resource scheduling; rather they rely on the user to select resources or services. Among such workflow systems are Kepler, Taverna, and VisTrails. In the case of Taverna, the user can provide a set of services that match a particular workflow component, so if errors occur, an alternate service can be automatically invoked. Some workflow management systems, such as P-GRADE [56] use an external broker to schedule tasks onto resources. The Askalon system [57], designed to support task-level workflows, has a rich environment for mapping workflows onto resources. It not only does the resource assignment but can also automatically provision the resources ahead of the workflow execution. Askalon contains two major components responsible for workflow scheduling: the scheduler and the resource management system (GridARM). GridARM serves as a data repository which provides the scheduler with all the information needed for scheduling, including the available resources and the applications deployed

on the Grid. Apart from the basic functionalities, GridARM can also provide more advanced resource and application discovery techniques based on quality-of-service matching, and it can guarantee advance reservation of resources. In order to support the scheduler, Askalon has developed a performance analysis and prediction system that can estimate the runtimes of the workflow tasks as well as times of data transfers between tasks. The scheduler performs full-graph scheduling of scientific workflows. Askalon can also support workflow transformation techniques to improve performance. However, unlike our approach, Askalon focuses on simplifying the conditional structures present in its workflows. Among the optimization transformations are techniques employed in parallel compilers, which include branch prediction, parallel loop unrolling, and sequential loop elimination. If the choices made during the transformation (such as branch prediction) are erroneous, the workflow is adjusted and rescheduled at runtime. Once the DAG is created, it is mapped onto the available resources based on a scheduling algorithm.

9. Conclusions

In this paper, we described the challenges faced by workflow management and workflow performance evaluation systems when supporting large-scale applications. We showed how a collaboration between domain scientists and computer scientists can lead to advances in both areas, driving the science forward as well as supporting the development of new computer science techniques and approaches.

We focused the paper on one particular application, CyberShake, which uses workflow technologies to produce a new and important type of probabilistic seismic hazard curve. The goal of CyberShake is to use full 3D waveform modeling on a regular basis within the seismological community. Showing that CyberShake is a practical technology, despite its complexity and computational challenges, is an important step towards that goal.

We described how the challenges in scaling up the workflow description can be met by developing support for recursive workflow definitions at the intermediate workflow representation level. With these definitions, we can not only reach the needed scale of almost a million tasks, but also support the mapping of tasks to resources close to the task execution time.

We described how various optimization techniques, including resource provisioning, task clustering and task release strategies can have a positive impact on performance.

Finally, we showed how scalable log mining techniques can help analyze the performance of large-scale workflows, providing a graphical view of the overall workflow performance from various points of view: at a task-level, cluster-level or workflow-level, as well at the resource utilization level. The log analysis also enabled us to troubleshoot the causes of unexpected overheads and thus avoid them in the future.

As a result of our work, through the process of designing, executing, and analyzing CyberShake, we made a series of optimizations enabling us to run end-to-end workflows of over 800,000 jobs in 18 hours on 800 processors. Over the course of three weeks, a single SCEC scientist was able to execute over 7.5 million jobs using grid computing. The improvements suggested in this paper are also valid and useful beyond CyberShake, as they are applicable to a wide range of embarrassingly parallel, high throughput computing applications.

Although we achieved the desired scalability and made progress in a number of areas, there are still many issues to be addressed. For example, in the future we plan to extend our current log analysis capabilities to online analysis that follows the progress of the workflow, its performance, and reports any problems. We also plan to provide additional troubleshooting capabilities to be able to zero-in on any workflow failures and provide enough workflow-level and system-level information so that the scientist will be able to easily track down sources of failures.

We have seen that choosing the right workflow-level and system-level parameters can improve the performance of the overall workflow. However, this tuning is currently done manually, with the scientists picking parameters in an ad-hoc fashion. In the future, we plan to develop techniques that would make these decisions in an automated way.

Acknowledgments

This work was partially supported by the National Center for Supercomputing Applications under TG-MCA03S012 (SCEC PetaScale Research: An Earthquake System Science Approach to Physics-based Seismic Hazard Research) and utilized NCSA's Mercury cluster. This work was supported in part by the Center for High Performance Computing and Communications at the University of Southern California. This work was also support by NSF under OCI-*0722019* and OCI-0749313.

References

[1] E. Deelman, D. Gannon, M. Shields, I. Taylor, Workflows and e-science: An overview of workflow system features and capabilities, Future Generation Computer Systems, doi:10.1016/j.future.2008.06.012, 2008.
[2] I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.
[3] B. Berriman, A. Bergou, E. Deelman, J. Good, J. Jacob, D. Katz, C. Kesselman, A. Laity, G. Singh, M.-H. Su, R. Williams, Montage: A grid-enabled image mosaic service for the NVO, in: Astronomical Data Analysis Software & Systems (ADASS), vol. XIII, 2003.
[4] G.B. Berriman, E. Deelman, J. Good, J. Jacob, D.S. Katz, C. Kesselman, A. Laity, T.A. Prince, G. Singh, M.-H. Su, Montage: A grid enabled engine for delivering custom science-grade mosaics on demand, in: SPIE Conference 5487: Astronomical Telescopes, 2004.
[5] G.B. Berriman, Ewa Deelman, J. Good, J.C. Jacob, D.S. Katz, A.C. Laity, T.A. Prince, G. Singh, M.-H. Su, Generating Complex Astronomy Workflows, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.
[6] I. Taylor, M. Shields, I. Wang, R. Philp, Distributed P2P computing within Triana: A Galaxy visualization test case, in: IPDPS 2003, 2003.

[7] T. Oinn, P. Li, D.B. Kell, C. Goble, A. Goderis, M. Greenwood, D. Hull, R. Stevens, D. Turi, J. Zhao, Taverna/myGrid: Aligning a workflow system with the life sciences community, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.
[8] R.D. Stevens, A.J. Robinson, C.A. Goble, myGrid: Personalised bioinformatics on the information grid, in: Eleventh International Conference on Intelligent Systems for Molecular Biology, Bioinformatics 19 (2003) 302–304.
[9] S. Callaghan, P. Maechling, E. Deelman, K. Vahi, G. Mehta, G. Juve, K. Milner, R. Graves, E. Field, D. Okaya, D. Gunter, K. Beattie, T. Jordan, Reducing time-to-solution using distributed high-throughput mega-workflows – Experiences from SCEC CyberShake, in: Fourth IEEE International Conference on e-Science (e-Science 2008), Indianapolis, IN, USA, 2008.
[10] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T.H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, L. Zhao, Managing large-scale workflow execution from resource provisioning to provenance tracking: The CyberShake example, in: E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, 2006, p. 14.
[11] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman, S. Callaghan, D. Okaya, H. Francoeur, V. Gupta, Y. Cui, K. Vahi, T. Jordan, E. Field, SCEC CyberShake workflows – Automating probabilistic seismic hazard analysis calculations, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.
[12] D.A. Brown, P.R. Brady, A. Dietz, J. Cao, B. Johnson, J. McNabb, A case study on the use of workflow technologies for scientific analysis: Gravitational wave data analysis, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.
[13] L. Piccoli, Lattice QCD workflows: A case study, in: SWBES08: Challenging Issues in Workflow Applications, Indianapolis, IN, 2008.
[14] M. Jones, B. Ludaescher, D. Pennington, A. Rajasekar, Data integration and workflow solutions for ecology, in: Data Integration in Life Sciences, 2005.
[15] Open Science Grid, http://www.opensciencegrid.org.
[16] TeraGrid, http://www.teragrid.org/.
[17] Enabling Grids for E-sciencE (EGEE), http://www.eu-egee.org/.
[18] Southern California Earthquake Center (SCEC), http://www.scec.org/, 2006.
[19] L. Zhao, P. Chen, T.H. Jordan, Strain Green's tensors, reciprocity, and their applications to seismic source and structure studies, Bull. Seismol. Soc. Amer. 96 (2006) 1753–1763.
[20] 2007 Working Group on California Earthquake Probabilities, The Uniform California Earthquake Rupture Forecast, Version 2 (UCERF 2): U.S. Geological Survey Open-File Report 2007-1437 and California Geological Survey Special Report 203, http://pubs.usgs.gov/of/2007/1437/, 2008.
[21] Pegasus, http://pegasus.isi.edu.
[22] E. Deelman, M. Livny, G. Mehta, A. Pavlo, G. Singh, M.-H. Su, K. Vahi, R.K. Wenger, Pegasus and DAGMan from concept to execution: Mapping scientific workflows onto today's cyberinfrastructure, in: L. Grandinetti (Ed.), High Performance Computing (HPC) and Grids in Action, IOS Press, 2008.
[23] E. Deelman, G. Mehta, G. Singh, M.-H. Su, K. Vahi, Pegasus: Mapping large-scale workflows to distributed resources, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.
[24] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G.B. Berriman, J. Good, A. Laity, J.C. Jacob, D.S. Katz, Pegasus: A framework for mapping complex scientific workflows onto distributed systems, Scientific Programming J. 13 (2005) 219–237.
[25] DAGMan, http://www.cs.wisc.edu/condor/dagman.
[26] R. Henderson, D. Tweten, Portable Batch System: External Reference Specification, 1996.
[27] S. Zhou, LSF: Load sharing in large-scale heterogeneous distributed systems, in: Workshop on Cluster Computing, 1992.
[28] M.J. Litzkow, M. Livny, M.W. Mutka, Condor – a hunter of idle workstations, in: Proc. 8th Int. Conf. on Distributed Computin Systems, 13–17 June 1988, pp. 104–111.
[29] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke, Security for grid services, in: Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), 2003.
[30] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripenu, B. Schwartzkopf, H. Stockinger, K. Stockinger, B. Tierney, Giggle: A framework for constructing scalable replica location services, in: Proceedings of Supercomputing 2002 (SC2002), 2002.
[31] E. Deelman, C. Kesselman, G. Mehta, Transformation catalog design for GriPhyN, Technical report GriPhyN-2001-17, 2001.
[32] J. Voeckler, G. Mehta, Y. Zhao, E. Deelman, M. Wilde, Kickstarting remote applications, in: GCE06 Second International Workshop on Grid Computing Environments, Tampa, Florida, 2006.
[33] B. Tierney, D. Gunter, NetLogger: A toolkit for distributed system performance tuning and debugging, in: Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management, 2003.
[34] R. Ihaka, R. Gentleman, R: A language for data analysis and graphics, J. Comput. Graph. Statist. (1996) 299–314.
[35] Grid Logging: Best Practices Guide, http://www.cedps.net/index.php/LoggingBestPractices, 2008.
[36] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, Concurrency and Computation: Practice & Experience, Special Issue: Workflow in Grid Systems (2005).
[37] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, T. Oinn, Taverna: A tool for building and running workflows of services, Nucleic Acids Res. 34 (April 2006) 729–732.
[38] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, S. Weerawarana, Specification: Business process execution language for web services Version 1.1, 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
[39] A. Slominski, Adapting BPEL to scientific workflows, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.
[40] Pegasus, Generating abstract workflows for Pegasus Planner, http://pegasus.isi.edu/mapper/docs/guides/ch0/index.html, 2009.
[41] condor_glidein, http://www.cs.wisc.edu/condor/glidein.
[42] G. Juve, E. Deelman, Resource provisioning options for large-scale scientific workflows, in: Third International Workshop on Scientific Workflows and Business Workflow Standards in e-Science (SWBES) in conjunction with Fourth IEEE International Conference on e-Science (e-Science 2008) Indianapolis, Indiana, USA, 2008.
[43] G. Singh, M.H. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D.S. Katz, G. Mehta, Workflow task clustering for best effort systems with Pegasus, in: Proceedings of the 15th ACM Mardi Gras Conference: From Lightweight Mash-Ups to Lambda Grids: Understanding the Spectrum of Distributed Computing Requirements, Applications, Tools, Infrastructures, Interoperability, and the Incremental Adoption of Key Capabilities, 2008.
[44] R. Graves, Physics based probabilistic seismic hazard calculations for Southern California, in: 14th World Conference on Earthquake, Engineering, Beijing, China, 2008.
[45] J. Yu, R. Buyya, A taxonomy of workflow management systems for grid computing, J. Grid Comput. 3 (2005) 171–200.
[46] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, K. Vahi, Characterization of scientific workflows, in: 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS08), 2008.
[47] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, J. Myers, Examining the challenges of scientific workflows, IEEE Comput. 40 (2007) 24–32.
[48] L. Ramakrishnan, Y. Simmhan, B. Plale, Realization of dynamically adaptive weather analysis and forecasting in LEAD: Four years down the road, IPAW 4487 (2007) 1122.
[49] J. Pahwa, R. White, A. Jones, M. Burgess, W. Gray, N. Fiddian, T. Sutton, P. Brewer, C. Yesson, N. Caithness, Accessing biodiversity resources in computational environments from workflow applications, in: WORKS06, 2006.

[50] Active BPEL Engine, http://www.active-endpoints.com/active-bpel-engine-overview.htm, 2007.
[51] Microsoft Workflow Foundation, http://msdn2.microsoft.com/en-us/netframework/aa663328.aspx, 2007.
[52] S. Miles, J. Papay, C. Wroe, P. Lord, C. Goble, L. Moreau, Semantic description, publication and discovery of workflows in myGrid, Electronics and Computer Science, University of Southampton, Technical report ECSTR-IAM04-001, 2004.
[53] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, I. Wang, Programming scientific and distributed workflow with Triana services, Concurrency and Computation: Practice and Experience 18 (2006) 1021–1037.
[54] S.P. Callahan, J. Freire, E. Santos, C.E. Scheidegger, C.T. Silva, H.T. Vo, VisTrails: Visualization meets Data Management, in: ACM SIGMOD, 2006.
[55] G. von Laszewski, M. Hategan, Workflow concepts of the Java CoG Kit, J. Grid Comput. 3 (2006) 239–258.
[56] A. Kertész, G. Sipos, P. Kacsuk, Brokering multi-grid workflows in the P-GRADE portal, in: Euro-Par 2006: Parallel Processing, vol. 4375, Springer, Berlin, 2006.
[57] M. Wieczorek, R. Prodan, T. Fahringer, Scheduling of scientific workflows in the ASKALON grid environment, SIGMOD Record 34 (2005).